University of Cyprus
Computer Science Department
Homework 2: RESTful API Java Client using the Jersey Framework
EPL344: Internet Technologies
Spring 2020

Announced Date: Thursday, 13/02/2020
Submission Date: Thursday, 05/03/2020 (23:59)

I. Introduction

The goal of this exercise is to write a RESTful API client program in Java through which you will obtain hands-on experience in HTTP request/response message exchange. Specifically, you are to implement an HTTP message exchange functionality between your program and some RESTful APIs over the Internet. You will use the Jersey RESTful Web Services framework which is an open source, production quality, framework for developing RESTful Web Services in Java. It is strongly advised that your Jersey project is built using Apache Maven software which is project build and management tool. You can find an example of a Maven-oriented Jersey-based RESTful API client in Lab5.

II. Goal

The goal of this assignment is to obtain, use and print:

(a) weather data via the OpenWeatherMap (https://openweathermap.org/) RESTful API

(b) routing data via the Graphhopper (https://www.graphhopper.com) RESTful Routing API

OpenWeatherMap provides a RESTful API service to provide current weather, daily forecast for 16 days, and 3-hourly forecast 5 days for any city on Earth by proving any of the following inputs: city name, city id\(^1\), geographic coordinates or zip code. Graphhopper provides a RESTful Direction API which consists of 6 APIs. More specifically, we will use the Routing API which returns the best path(s) between two or more points.

In this assignment, you will combine weather forecast data from OpenWeatherMap API and navigation data between two points from Graphhopper Routing API to develop a web service.

\(^1\) List of city ID city.list.json.gz can be downloaded here http://bulk.openweathermap.org/sample/
III. Prerequisites

To be able to carry out this assignment you will need to:

- **Sign up** for a free account with OpenWeatherMap. Once you register, you will receive an **APPID**, essentially your key for using the service. Without an APPID you will not be able to access any API endpoint. It takes up to 1 hour to activate your API key. You will receive a confirmation email as your API key is ready to work. The How to start page tells how to include the APIID and id in a request. Unless you care to convert from Kelvin to degrees Celsius in your code, be sure to request ‘metric’ units from the server.
- **Register** for a free account with Graphhopper. Once you register, you will receive an **API KEY**, which is required for using the service.

IV. Description of Work

Your java program will offer 4 different options according to given command-line arguments:

1. Return the current weather conditions for a **given location**
2. Return 5-days weather forecast for a **given location**
3. Return routing instructions for travelling by car from a **given starting point** to a **given destination point**
4. Return information about an excursion you plan to do within the next 5-days from a **given starting point** to a **given destination**, given the fact that you want to arrive at the destination when the highest temperature (throughout the whole duration) is reached.

Every option involves one or more requests to appropriate API endpoint(s) from OpenWeatherMap and/or Graphhopper APIs. The next section analyzes all API endpoints involved in this assignment. The requests from you API java client will advertise that the client accepts JSON data format. Upon response arrival, your client will parse the JSON obtained within the body of the response using Gson\(^2\) json parser. In order to use GSON, place the following dependency in your pom.xml file:

```xml
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.8.5</version>
</dependency>
```

For the sake of further clarifying and guiding the development process, an **as2-supplementary.zip** file is given that includes a full Maven project (OpenWeatherMap folder) with a properly set pom.xml file (with all dependencies needed for the assignment\(^3\)) as well as source code .java files which implement the first option (returns the current weather conditions). You will have to study the given solution and build the rest options using the same approach. You may unzip the folder, get into OpenWeatherMap folder and run “mvn clean package”.

---

\(^2\) Provides simple toJson() and fromJson() methods to convert Java objects to JSON and vice-versa.

\(^3\) You can add mode dependencies in case you want to use any other libraries.
V. Involved API Endpoints

The API endpoints will be used in this assignment are shown below. The first 2 endpoints are related to OpenWeatherMap API and the third endpoint relates to Graphhopper API.

1. **Current weather**: http://api.openweathermap.org/data/2.5/weather

Example of API endpoint with parameters:

http://api.openweathermap.org/data/2.5/weather?q={city_name},{country_code}&units=metric&APPID={APPID}

**Parameters:**

- **q**: city_name and country_code divided by comma, use ISO 3166 country codes
- **units**: metric (temperature in Celsius, distance in kilometers), imperial (temperature in Fahrenheit, distance in miles). When you do not use units parameter, format is Standard (temperature in Kelvin, distance in kilometers) by default.

**Example of API call that returns the current weather information for Nicosia, cy**

http://api.openweathermap.org/data/2.5/weather?q=Nicosia, cy&units=metric&APPID={APPID}

The following request headers and reply headers and body were obtained using the curl command from command line in order to study the messages exchanged. For security reasons, the APPID was replaced by XXXXX from the information below. The curl command syntax is:

curl -v "http://api.openweathermap.org/data/2.5/weather?q=Nicosia, cy&units=metric&APPID=XXXXX"

**Request**

GET /data/2.5/weather?q=Nicosia, cy&units=metric&APPID=XXXXX HTTP/1.1
Host: api.openweathermap.org
User-Agent: curl/7.58.0
Accept: */*

**Reply**

HTTP/1.1 200 OK
Server: openresty
Date: Thu, 14 Feb 2019 08:46:32 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 452
Connection: keep-alive
X-Cache-Key: /data/2.5/weather? APPID=XXXXX&q=nicosia, cy&units=metric
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET, POST
The json obtained in 200 OK response body can be beautified using online json validators such as https://jsonlint.com/ as follows:

```json
{
    "coord": {
        "lon": 33.37,
        "lat": 35.17
    },
    "weather": [{
        "id": 802,
        "main": "Clouds",
        "description": "scattered clouds",
        "icon": "03n"
    }],
    "base": "stations",
    "main": {
        "temp": 3.93,
        "feels_like": -0.53,
        "temp_min": 3,
        "temp_max": 4.44,
        "pressure": 1028,
        "humidity": 51
    },
    "visibility": 10000,
    "wind": {
        "speed": 2.6,
        "deg": 360
    },
    "clouds": {
        "all": 40
    },
    "dt": 1581274882,
    "sys": {
        "type": 1,
        "id": 6370,
        "country": "CY",
        "sunrise": 1581223178,
        "sunset": 1581261713
    },
    "timezone": 7200,
    "id": 146268,
    "name": "Nicosia",
    "cod": 200
}
```
2. **3-hourly 5-days Weather Forecast**: http://api.openweathermap.org/data/2.5/forecast

Example of API endpoint with parameters:

http://api.openweathermap.org/data/2.5/forecast?q={city_ID},{country_code}&units=metric&APPID={APPID}

**Parameters:**

- **q**: city name and country code divided by comma, use ISO 3166 country codes
- **units**: metric (temperature in Celsius, distance in kilometers), imperial (temperature in Fahrenheit, distance in miles). When you do not use units parameter, format is Standard (temperature in Kelvin, distance in kilometers) by default.

**Example of API call that returns the 5-day historical weather information for Nicosia,cy**

http://api.openweathermap.org/data/2.5/forecast?q=Nicosia,cy&units=metric&APPID={APPID}

The following request headers and reply headers and body were obtained using the curl command from command line in order to study the messages exchanged. The curl command syntax is:

```bash
curl -v "http://api.openweathermap.org/data/2.5/forecast?q=Nicosia,cy&units=metric&APPID=XXXXX"
```

**Request**

GET /data/2.5/forecast?q=Nicosia,cy&units=metric&APPID=XXXXX HTTP/1.1
Host: api.openweathermap.org
User-Agent: curl/7.58.0
Accept: */*

**Reply**

HTTP/1.1 200 OK
Server: openresty
Date: Sun, 09 Feb 2020 19:11:27 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 14261
Connection: keep-alive
X-Cache-Key: /data/2.5/forecast?APPID=XXXXX&q=nicosia,cy&units=metric
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET, POST

```json
```
From the body of 40 (see cnt value in json) 3-hourly forecasts above, we show only the first two forecasts. In order to better preview the json returned from the weather forecast api endpoint you may use the json validator [https://jsonlint.com/](https://jsonlint.com/).

### 3. Finding the best paths between two or more points: [https://graphhopper.com/api/1/route](https://graphhopper.com/api/1/route)

Example of API endpoint with parameters:

https://graphhopper.com/api/1/route?point=lat,lon&point=lat,lon&vehicle=type&key=API_KEY

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>point</td>
<td>-</td>
<td>Specify multiple points for which the route should be calculated. The order &quot;latitude,longitude&quot; is important. Specify at least two points. The maximum number depends on the selected package.</td>
</tr>
<tr>
<td>locale</td>
<td>en</td>
<td>The locale of the resulting turn instructions. E.g. el_GR for Greek or de for German.</td>
</tr>
<tr>
<td>optimize</td>
<td>false</td>
<td>If false the order of the locations will be identical to the order of the point parameters. If you have more than 2 points you can set this optimize parameter to true and the points will be sorted regarding the minimum overall time - e.g. suitable for sightseeing tours or salesman. Keep in mind that the location limit of the Route Optimization API applies and the credit costs are higher! Note to all customers with a self-hosted license: this parameter is only available if your package includes the Route Optimization API.</td>
</tr>
<tr>
<td>instructions</td>
<td>true</td>
<td>If instruction should be calculated and returned</td>
</tr>
<tr>
<td>Parameter</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>vehicle</td>
<td>car</td>
<td>The vehicle for which the route should be calculated. Other vehicle profiles are listed <a href="#">here</a>.</td>
</tr>
<tr>
<td>elevation</td>
<td>false</td>
<td>If true a third dimension - the elevation - is included in the polyline or in the GeoJson. IMPORTANT: If enabled you have to use a modified version of the decoding method or set points_encoded to false. See the points_encoded attribute for more details. Additionally a request can fail if the vehicle does not support elevation. See the features object for every vehicle.</td>
</tr>
<tr>
<td>points_encoded</td>
<td>true</td>
<td>If false the coordinates in point and snapped_waypoints are returned as array using the order [lon,lat,elevation] for every point. If true the coordinates will be encoded as string leading to less bandwidth usage. You'll need a special handling for the decoding of this string on the client-side. We provide open source code code in Java and JavaScript, see the clients section. It is especially important to use our official client or code if you set elevation=true!</td>
</tr>
<tr>
<td>calc_points</td>
<td>true</td>
<td>If the points for the route should be calculated at all printing out only distance and time.</td>
</tr>
<tr>
<td>debug</td>
<td>false</td>
<td>If true, the output will be formated.</td>
</tr>
<tr>
<td>type</td>
<td>json</td>
<td>Specifies the resulting format of the route, for json the content type will be application/json. Or use gpx, the content type will be application/gpx+xml, see below for more parameters.</td>
</tr>
<tr>
<td>point_hint</td>
<td>-</td>
<td>Optional parameter. Specifies a hint for each point parameter to prefer a certain street for the closest location lookup. E.g. if there is an address or house with two or more neighboring streets you can control for which street the closest location is looked up.</td>
</tr>
</tbody>
</table>
| details       | -       | Optional parameter to retrieve path details. You can request additional details for the route: street_name and time. For all
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>motor vehicles we additionally support</td>
<td></td>
<td>max_speed, toll (no, all, hgv), road_class (motorway, primary, ...), road_environment, and surface. The returned format for one details is [fromRef, toRef, value]. The ref references the points of the response. Multiple details are possible via multiple key value pairs details=time&amp;details=toll.</td>
</tr>
</tbody>
</table>

**Example of API endpoint that returns the routing information for travelling from Nicosia,cy to Paphos,cy by car**

https://graphhopper.com/api/1/route?point=35.1667,33.3667&point=34.7667,32.4167&vehicle=car&key={API_KEY}

The following request headers and reply headers and body were obtained using the curl command from command line in order to study the messages exchanged. The curl command syntax is:

```
curl -v "https://graphhopper.com/api/1/route?point=35.1667,33.3667&point=34.7667,32.4167&vehicle=car&key=XXXXX"
```

**Request**

```
GET /api/1/route?point=35.1667,33.3667&point=34.7667,32.4167&vehicle=car&key=XXXXX HTTP/1.1
Host: graphhopper.com
User-Agent: curl/7.58.0
Accept: */*
```

**Reply**

```
HTTP/2 200
server: nginx
date: Sun, 09 Feb 2020 19:18:26 GMT
content-type: application/json;charset=utf-8
content-length: 6410
access-control-expose-headers: X-RateLimit-Limit,X-RateLimit-Remaining,X-RateLimit-Reset,X-RateLimit-Credits
access-control-allow-origin: *
x-ratelimit-limit: 500
x-ratelimit-remaining: 500
x-ratelimit-reset: 16894
x-ratelimit-credits: 1
strict-transport-security: max-age=31536000; includeSubDomains
```
The output format and values of the above API endpoint is described [here](#). For example, distance is measured in meters and time (duration) of the route in milliseconds.

**VI. Examples of Execution**

In this section, we provide execution examples for the 4 program options. Your program must return the same output for each option as the examples shown below. Please make sure that you print appropriate error messages: (a) if the number of arguments is invalid, (b) if the latitude (-90 to +90) and longitude (-180 to +180) values are within the limits (related to option 3) and (c) if the name of the location is invalid. The latter problem will emerge when you send a request to OpenWeatherMap and you receive a 404 Not Found response. Error messages will be taken into account in the assignment grade.

**OPTION 1:** The output of the first option is the beautified value of the section “weather” (the first element of the list) and “main” of the json response as shown below. The source code for that option is given. Thus, you do not have to re-implement it but you have to incorporate it in your Maven project.
OPTION 2: The output of the second option is the beautified value of the first element of the forecast list within the “list” section of the JSON response as shown below:

```java
java -cp <Classpath> <packages>.<class> -weatherForecast Nicosia,cy
{
    "dt": 1550145600,
    "main": {
        "temp": 14.36,
        "temp_min": 12.44,
        "temp_max": 14.36,
        "pressure": 1013.41,
        "sea_level": 1022.54,
        "grnd_level": 1013.41,
        "humidity": 100,
        "temp_kf": 1.92
    },
    "weather": [{
        "id": 501,
        "main": "Rain",
        "description": "moderate rain",
        "icon": "10d"
    }],
    "clouds": {
        "all": 80
    },
    "wind": {
        "speed": 1.27,
        "deg": 285.006
    },
    "rain": {
        "3h": 3.4375
    },
    "sys": {
        "pod": "d"
    },
    "dt_txt": "2020-02-09 12:00:00"
}
```
OPTION 3: The output of the third option is the beautified value of the first two elements of the instructions list within the “instructions” section of the json response as shown below. The four command line arguments after the -routing option are: source latitude, source longitude, destination latitude, and destination longitude respectively.

```java
java -cp <Classpath> <packages>.<class> -routing 35.1667 33.3667 34.7667 32.4167
```

```
{
  "distance": 23.789,
  "heading": 254.97,
  "sign": 0,
  "interval": [0, 1],
  "text": "Continue onto Pindarou",
  "time": 3058,
  "street_name": "Pindarou"
}
{
  "distance": 220.244,
  "sign": -2,
  "interval": [1, 3],
  "text": "Turn left onto Androkleous Street",
  "time": 49554,
  "street_name": "Androkleous Street"
}
```

OPTION 4: The output of the fourth option is information about an excursion you plan to do within the next 5-days from a given starting point to a given destination, given the fact that you want to arrive at the destination when the highest temperature (throughout the whole duration) is reached. When searching for the highest temperature in the next 5 days, use the “temp” field in “main” section of the OpenStreetMap response, and the dt_txt field. The geographic coordinations (latitude, longitude) of both locations will be found on weather forecast responses.

```java
java -cp <Classpath> <packages>.<class> -excursion -from Nicosia,cy -to Paphos,cy
```

Highest temperature at destination is 15.69 on 2019-02-19 12:00:00
Trip distance: 149.581 km
Time to reach destination: 104.49 minutes

**Important Notes for OpenWeatherMap API**

- In the free plan, there is a [limit](https://openweathermap.org) of 60 requests per minute. If you do not receive a response from the API, please, wait at least for 10 min and then repeat your request.
- The server name is [api.openweathermap.org](https://openweathermap.org). Please, never use the IP address of the server.

**Important Notes for Graphhopper Routing API**

- In the free plan, there is a [limit](https://graphhopper.com) of 500 requests per day and up to 5 locations per request. The latter restriction is out of the scope of this assignment, since you will use the routing API with two locations only (start, stop).
VII. Submission

Run your program three times, one for obtaining the weather forecast, one for obtaining the routing information and one for running scenario of option 4 and put all the commands along with the results (output) of your program in a document. For every run related to OpenWeatherMap API, please provide a wireshark screenshot providing the involved request/response pairs as shown in slides 37-38 of Lab5 which will illustrate at least the headers of messages exchanged (e.g. for the weather forecast and routing cases, the body of the response messages will not be completely depicted due to its long format).

Create a .zip file that will contain:

1. your whole Maven project (the folder created by Maven). Before you zip please run the command “mvn clean” to remove target folder containing large .jar files and minimize Maven project folder size.
2. your document as described above

and submit it to Moodle.

APPENDIX

In rain section in OpenStreetMap weather forecast, there is a json string 3h starting with number. We cannot define a variable name starting with number so we can use annotations:

```java
import com.google.gson.annotations.SerializedName;

public class Rain {
    @SerializedName(value = "3h")
    private float threeHours;
}
```

4 The communication with OpenWeatherMap is done over the http protocol thus request/response messages appear in wireshark. On the other hand, the communication with Graphhopper is done over the https. Therefore, all packets exchanged are encrypted and request/response messages cannot be identified and printed.